POLYTECH SORBONNE

Sorbonne Université
2020-2021

Cours de Programmation en C – EI-2I

Travaux Pratiques

Objectif(s)

- Allocation dynamique de mémoire
- Désallocation
- Chaînes de caractères

Exercice 1 – Fonction `strcmp`

On cherche à ré-écrire le code de la fonction `strcmp` qui compare les chaînes `s1` et `s2` suivant l'ordre lexicographique[1]. Le résultat est négatif si `s1<s2`, 0 si `s1==s2` et positif si `s1>s2`.
Le prototype de la fonction est `int strcmp( char* s1, char* s2)`.
Dans un même fichier `strcmp.c` :

1. Écrire une fonction `strcmp1` qui réalise la comparaison de deux chaînes en utilisant l'opérateur d'indexation `[ ]`.
2. Écrire une fonction `strcmp2` qui réalise aussi la comparaison, mais en utilisant exclusivement des pointeurs que l'on incrémente.
3. Écrire un programme principal qui saisit deux chaînes, puis affiche le résultat de la comparaison, en utilisant les fonctions `strcmp1` et `strcmp2` et la vraie fonction `strcmp`.

Rappel : il est possible de comparer deux caractères (`char`) de la même manière que l'on compare des entiers (si `a` et `b` sont des variables de type `char`, alors le code `a<b` renvoie 1 si `a` est plus petit (dans le sens alphabétique) que `b`, et 0 sinon). De plus, on **n'**utilisera **pas** de fonctions de la bibliothèque standard `string` pour écrire `strcmp1` et `strcmp2`.

Exercice 2 – Recherche de séquence ADN

On cherche, dans cet exercice, à mener une enquête : un fragment d'ADN a été retrouvé dans un camping-car suspecté d'avoir abrité un laboratoire illégal de fabrication de méthamphétamine. Parmi un fichier fictif [2] d'ADN, on cherchera la personne dont le fragment d'ADN donné présente le plus de ressemblance avec l'ADN trouvée sur une preuve.

On va mettre en place une *mesure* (au sens mathématique) indiquant la *similarité* entre deux séquences ADN, et on s'en servira pour trouver les séquences les plus proches (au sens de cette mesure) d'une séquence donnée.

Dans ce TP, toutes les chaînes de caractères manipulées n'ont pas de taille *a priori*. Il faudra donc utiliser des pointeurs de caractères (`char*`) et l'allocation dynamique de mémoire. N'oubliez donc pas d'**allouer** et de **désallouer** ! Les fonctions de manipulations de chaînes de caractères (`strcpy`, `strlen`, `strcat`, `strcmp`, etc.) seront aussi utilisées.

[1] L'ordre lexicographique est l'ordre utilisé dans les dictionnaires. Il s'agit de l'ordre alphabétique pour la 1ère lettre, puis si elles sont égales, de l'ordre alphabétique pour la 2ème, puis pour la 3ème, etc.

[2] Pour rappel, tout fichier d'individus est strictement contrôlé par la loi et doit faire l'objet d'une déclaration à la CNIL : `http://www.cnil.fr` et `http://www.cnil.fr/vos-responsabilites/declarer-a-la-cnil/`. Il ne s'agit ici que d'un prétexte pour un TP, la création d'un tel fichier étant éthiquement discutable.

1 Données personnelles

Dans le fichier ADN, chaque personne est renseignée par les champs suivants :

- Nom
- Prenom
- séquence ADN

où une séquence ADN est une chaîne de caractères composée uniquement des caractères "A" (pour Adenosine), "C" (pour Cytosine), "G" (pour Guanine) et "T" (pour Thymidine). Le caractère "–" sera aussi utilisé pour un nucléotide que l'on n'aura pas pu déterminer.

Dans des fichiers `individu.h` et `individu.c` :

Question 1
Créer une structure de données `t_indiv` pour stocker un individu (on ne stockera que le nom et l'ADN).

Question 2
Créer une fonction qui reçoit en paramètres un pointeur sur un fichier et un pointeur sur un individu et qui remplit cet individu avec le nom et une séquence ADN lue dans le fichier (le nom sera composé du nom lu, d'un espace et du prénom lu. Les noms et prénoms lus ne feront pas sur plus de 50 caractères, tandis que l'ADN lue comportera moins de 10000 nucléotides; on allouera évidemment dans le `t_indiv` la bonne quantité de mémoire pour recopier les chaines lues).

Question 3
Créer une procédure qui affiche un `t_indiv` passé en paramètre. Il faudra afficher quelque chose du genre

```
Nom: Gilligan V.
ADN: TCATATTTGCAGCTACAGCGTTTAATTTGCGCATCGGACTTCTAGTTGCGAC-ATCCTAC
CCCACCCATATAGAGTTCAATGGGTAGTATCCGTAGCCTATAGGAGTCCG–CAC
```

Question 4
Créer une procédure qui détruit un `t_indiv` (qui désalloue toute la mémoire allouée à sa création). Avant de désallouer un pointeur (ici des `char*`), on vérifiera qu'ils ne pointent pas vers `NULL`.

Question 5
Dans un fichier `mainP1.c`, écrire un programme principal permettant de tester ces trois fonctions (par exemple en stockant les 10ers individus lus dans le fichier, en les affichant (dans l'ordre inverse) puis en désallouant la mémoire allouée).

Question 6
Écrire un fichier `Makefile` permettant de créer l'exécutable `P1.out`

2 Degré de ressemblance entre chaînes

L'algorithme de Needleman et Wunsch[3] mesure la similarité (alignements) entre deux chaînes de caractères.

Il renvoie un score (entier) qui est d'autant plus grand que le nombre de ressemblances entre les deux chaînes est grand. Si une chaîne `ch1` est inclue complètement dans une chaîne `ch2`, alors cet algorithme indique la taille de `ch1`. Si, à quelques modifications près, `ch1` est inclue dans `ch2`, alors le score est égal à la taille de `ch1` moins le nombre de modifications. Cet algorithme peut être utilisé, par exemple, en bio-informatique pour calculer la similarité entre des séquences d'ADN[4].

L'algorithme de Needleman et Wunsch est donné par l'algorithme 1.

Si l'on divise ce score par la longueur de la plus petite des deux chaînes, on obtient le **pourcentage d'inclusion** donné par

$$P = \frac{score(\texttt{ch1},\texttt{ch2})}{\min(longueur(\texttt{ch1}),longueur(\texttt{ch2}))}$$

[3] `http://fr.wikipedia.org/wiki/Algorithme_de_Needleman-Wunsch`
A general method applicable to the search for similarities in the amino acid sequence of two proteins, Journal of Molecular Biology, vol 48(3), March 1970, pp 443–453.

[4] D'autres algorithmes, plus rapides et compliqués, sont aussi utilisés, notamment la méthode BLAST : `http://en.wikipedia.org/wiki/BLAST`

Algorithme 1 : algorithme de Needleman et Wunsch

Entrées : `ch1`, `ch2` : chaînes de caractères

```
1  début
2      ℓ1 ← longueur(ch1) +1
3      ℓ2 ← longueur(ch2) +1
4      d : tableau de ℓ1*ℓ2 entiers
5      pour i←0 à ℓ1-1 faire
6          d[ i*ℓ2 ] ← 0
7      fin
8      pour  j←0 à ℓ2-1 faire
9          d[ j ] ← 0
10     fin
11     pour i←1 à ℓ1-1 faire
12         pour j←1 à ℓ2-1 faire
13             si ch1[i-1]=ch2[j-1] alors
14                 cout←1
15             sinon
16                 cout← −1
17             fin
18             d[ i*ℓ2 + j ] ← maximum( d[ (i-1)*ℓ2 +j ] - 1,
19                                      d[ i*ℓ2 + j - 1 ] - 1,
20                                      d[ (i-1)*ℓ2 + j - 1] + cout)
21         fin
22     fin
23     retourner la plus grande valeur de d
24 fin
```

Question 7
Écrire dans un fichier `score.c` la fonction qui utilise l'algorithme de Needleman et Wunsch pour calculer la similarité entre deux chaînes.

Question 8
En déduire une fonction qui calculer le pourcentage d'inclusion entre deux chaînes (le résultat du calcul est un réel entre 0 et 1).

Question 9
Écrire un programme principal (dans `mainP2.c`) qui calcule et affiche le score et le pourcentage d'inclusion entre deux chaînes de caractères saisies au clavier. Par exemple, entre "`abcdefgh`" et "`cdefg`", le score vaut 5 et le pourcentage 100% (car 5 caractères de "`cdefg`" sont inclus dans "`abcdefgh`"). Pour "`abcdefgh`" et "`cdexfg`", le score vaut seulement 4, car il y a le "`x`" en trop, et le pourcentage 66.6%.

3 Recherche dans le fichier

Étant donné le fragment incomplet `"TAAAGGCCG-TAAGTTCCAAATT"` retrouvé dans le laboratoire clandestin, le but est de vérifier si l'une des personnes dans le fichier peut être suspectée. Pour cela, nous allons comparer le fragment relevé aux fragments du fichier afin de déterminer le (probable) coupable.

Question 10
Enrichir la structure `t_indiv` avec un champ réel pour y stocker le pourcentage de similarité avec une chaîne donnée. Écrire une fonction `calculeSimilarite` (dans `individu.c`) qui met-à-jour un individu en calculant le pourcentage d'inclusion (similarité) de son ADN et d'une chaine.

Question 11
Écrire (dans un fichier `mainP3.c` qui reprend `mainP1.c`) un programme qui ouvre le fichier fourni `ADN.txt`, lit les informations individu par individu en remplissant un individu avec les données lues puis les supprime (on pourra utiliser la fonction `feof` pour savoir si on est arrivé en fin de fichier).

Question 12
Compléter ce programme pour qu'il saisisse une séquence ADN, et calcule la similarité entre chaque individu et la séquence entrée.

Question 13
Compléter ensuite pour ne garder que l'individu qui possède la plus grande ressemblance avec la séquence donnée en entrée (pour cela, il faut penser à avoir une variable individu pour l'individu en cours et une pour l'individu qui présente la plus grande similarité). Afficher l'individu suspect en fin de programme.

Tester le programme pour trouver qui aurait pu utiliser ce laboratoire clandestin.

4 Vérification de l'allocation/désallocation de mémoire

Comme vu en TP de OPI, testez votre programme avec `valgrind` pour tester que toute la mémoire allouée a bien été désallouée à la fin du programme. Ajouter en commentaires le résultat de la vérification.

5 Bonus : recherche des individus les plus probables

On cherche maintenant à garder les N individus les plus probables (ceux avec les plus grandes similarités avec la séquence initiale), où N est une valeur saisie par l'utilisateur en début de programme.
Pour cela, on se propose de créer et de mettre à jour un tableau de N `t_indiv`, contenant les N individus ayant la plus grande similarité avec la séquence initiale. Pour plus de facilité, ce tableau restera trié suivant la similarité des individus pointés (du plus similaire au moins similaire). Au départ, on remplira le tableau avec des noms pointant vers `NULL`.

Question 14
Écrire une fonction vérifiant si un individu donné a sa place dans le tableau (commencer par poser l'algorithme sur papier). Ne pas oublier de considérer les individus avec un nom `NULL` (pour les 1^{ers} individus du fichier)).

Question 15
Écrire une procédure permettant l'insertion de cet élément (et le décalage des éléments suivants, et la désallocation du dernier). Commencez par poser l'algorithme nécessaire sur papier.

Question 16
Compléter le programme principal (dans un fichier `mainP4.c`) pour garder dans un tableau les N individus les plus probables. En fin de lecture du fichier, les afficher.